Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 ПО ДИСЦИПЛИНЕ:

ТИПЫ И СТРУКТУРЫ ДАННЫХ

ТЕМА: ОБРАБОТКА ОЧЕРЕДЕЙ

Студент **Поздышев А. В.**

Группа **ИУ7-31Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Поздышев А. В.**

Преподаватель _____ **Барышникова М. Ю.**

2024

ОГЛАВЛЕНИЕ

ОПИСАНИЕ УСЛОВИЯ ЗАДАЧИ

Система массового обслуживания состоит из двух обслуживающих аппаратов (ОА1 и ОА2) и двух очередей заявок. Всего в системе обращается 100 заявок. Заявки поступают в "хвост" каждой очереди; в ОА они поступают из "головы" очереди по одной и обслуживаются по случайному закону за интервалы времени T1 и T2, равномерно распределенные от 0 до 6 и от 1 до 8 единиц времени соответственно. (Все времена – вещественного типа). Каждая заявка после ОА1 с вероятностью 1-P=0.7 вновь поступает в "хвост" первой очереди, совершая новый цикл обслуживания, а с вероятностью P входит во вторую очередь. В начале процесса все заявки находятся в первой очереди.

ОПИСАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

Входные данные

Очередь из 100 элементов типа double.

Величина вероятности успешности выполнения, $0 < p \leq 1$.

Интервалы времени работы двух аппаратов ОА1 и ОА2, $t1 < t2$ в (ед. в).

Кол-во элементов n в очереди (для замерного эксперимента).

Выходные данные

Общее время моделирования, время простоя ОА2, количество срабатываний ОА1, среднее времени пребывания заявок в очереди (ед. в.).

Время измерений скорости выполнения операций (мс.).

Способ обращения к программе

Пользователь через консоль запускает программу: ./app.exe. После чего программа выводит меню опций.

Меню программы:

1. Вывести очереди.
2. Запуск процесса моделирования для очереди.
3. Запуск процесса моделирования для списка.
4. Измерить время добавления/удаления n элементов для двух способов.
5. Добавить число в очередь массив.
6. Добавить число в очередь список.
7. Удалить элемент из очереди массива.
8. Удалить элемент из очереди списка.
9. Вывести освобожденные адреса.
10. Выход.

Программа ожидает ввод одной из этих опций.

Описание возможных аварийных ситуаций и ошибок пользователя

Аварийные ситуации:

- Пустое поле ввода (ожидание ввода пользователя);
- Переполнение: добавление элемента в заполненную очередь;
- Удаление: удаление элемента из пустой очереди;

Ошибки пользователя:

- Некорректный ввод: вероятность p вне интервала (0, 1];
- Некорректный ввод: неверный интервал времени $t1 \geq t2$;
- Некорректный ввод: отрицательное время $t < 0$;
- Некорректный ввод: кол-во итераций n отрицательное;

ОПИСАНИЕ ВНУТРЕННИХ СТРУКТУР ДАННЫХ

Для представления очереди массива используется структура queue_t.

Листинг структуры queue_t:

```
#define MAX_QUEUE_LEN 10000
typedef struct queue
{
   double massive[MAX_QUEUE_LEN];

   int pout;

   int pin;

} queue_t;
```

1. massive – массив вещественный чисел (элементы очереди).
2. pout – индекс позиции выхода из очереди.
3. pin – индекс позиции входа в очередь.

В качестве представления элемента очереди списка используется структура node_t.

Листинг структуры node_t:

```
typedef struct node
{
   double value;

   struct node *prev;

} node_t;
```

1. value – значение элемента типа double.
2. pin – указатель типа node_t на предыдущий элемент списка.

Для представления очереди списка используется структура list_t.

Листинг структуры queue_t:

```
typedef struct list
{
  node_t *pin;

  node_t *pout;

} list_t;
```

1. pin – указатель типа node_t на первый элемент.
2. pout – указатель типа node_t на последний элемент.

ФУНКЦИИ ПРОГРАММЫ

Init_queue

Заголовок:

```
int init_queue(queue_t *queue);
```

Функция инициализирует очередь массив. Принимает указатель на структуру queue_t. Возвращает код ошибки.

Add_to_queue

Заголовок:

```
int add_to_queue(queue_t *queue, const double value);
```

Функция добавляет элемент в очередь массив. Принимает указатель на структуру queue_t, значение добавляемого элемента value. Возвращает код ошибки.

Pop_frm_queue

Заголовок:

```
int pop_frm_queue(queue_t *queue);
```

Функция удаляет элемент из очереди массива. Принимает указатель на структуру queue_t. Возвращает код ошибки.

Take_frm_queue

Заголовок:

```
int take_frm_queue(queue_t *queue, double *taken);
```

Функция забирает элемент из очереди массива. Принимает указатель на структуру queue_t, указатель на элемент taken. Возвращает код ошибки.

Init_list

Заголовок:

```
int init_list(list_t *list);
```

Функция инициализирует очередь список. Принимает указатель на структуру list_t. Возвращает код ошибки.

Free_list

Заголовок:

```
void free_list(list_t *list);
```

Функция освобождает память, выделенную под очередь список. Принимает указатель на структуру list_t.

Add_to_list

Заголовок:

```
int add_to_list(list_t *list, const node_t *node);
```

Функция добавляет элемент в очередь список. Принимает указатель на структуру list_t, указатель типа node_t на элемент node. Возвращает код ошибки.

Pop_frm_list

Заголовок:

```
int pop_frm_list(list_t *list);
```

Функция удаляет элемент из очереди списка. Принимает указатель на структуру list_t. Возвращает код ошибки.

Take_frm_queue

Заголовок:

```
int take_frm_list(list_t *list, double *taken);
```

Функция забирает элемент из очереди списка. Принимает указатель на структуру list_t, указатель на элемент taken. Возвращает код ошибки.

Print_queue

Заголовок:

```
void print_queue(queue_t *queue);
```

Функция выводит очередь массив на экран. Принимает указатель на структуру queue_t.

Print_list

Заголовок:

```
void print_list(list_t *list);
```

Функция выводит очередь список на экран. Принимает указатель на структуру list_t.

Process_modeling_queue

Заголовок:

```
int process_modeling_list(double *complete_time, double *down_time, double *avg_time_queue, double po, double oa1t1, double oa1t2, double oa2t1, double oa2t2);
```

Функция осуществляет процесс моделирования для очереди массива. Принимает указатели на время выполнения complete_time, время простоя down_time, среднее время в очереди avg_time_queue, вероятность успешности po, интервалы времени для OA1 oa1t1 и oa1t2, OA2 oa2t1 и oa2t2. Возвращает код ошибки.

Process_modeling_list

Заголовок:

```
int process_modeling_list(double *complete_time, double *down_time, double *avg_time_queue, double po, double oa1t1, double oa1t2, double oa2t1, double oa2t2);
```

Функция осуществляет процесс моделирования для очереди списка. Принимает указатели на время выполнения complete_time, время простоя down_time, среднее время в очереди avg_time_queue, вероятность успешности po, интервалы времени для OA1 oa1t1 и oa1t2, OA2 oa2t1 и oa2t2. Возвращает код ошибки.

ОПИСАНИЕ АЛГОРИТМА

Алгоритм выводит меню действий и ожидает ввод пользователя.

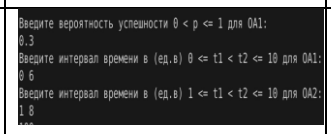
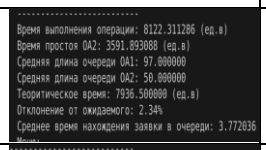
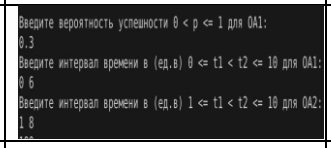
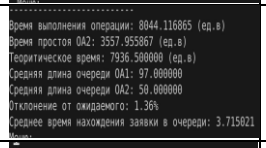
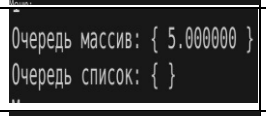
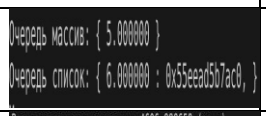
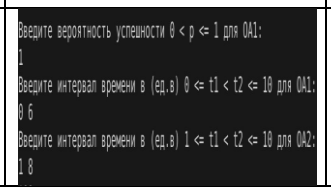
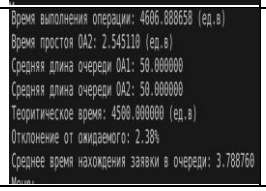
0. Действие 1:
    a. Вывод очереди массива.
    b. Вывод очереди списка.
1. Действие 2:
    a. Ввод значения вероятности.
    b. Ввод интервала времени для OA1.
    c. Ввод интервала времени для OA2.
    d. Запуск моделирования для очереди массива.
    e. Нахождение теоретического времени выполнения.
    f. Вывод результатов выполнения моделирования.
2. Действие 3:

a. Ввод значения вероятности.
b. Ввод интервала времени для ОА1.
c. Ввод интервала времени для ОА2.
d. Запуск моделирования для очереди списка.
e. Нахождение теоретического времени выполнения.
f. Вывод результатов выполнения моделирования.

3. Действие 4:
    a. Ввод кол-ва элементов в очереди.
    b. Замер времени добавления n элементов для двух способов.
    c. Замер времени удаления n элементов для двух способов.
    d. Вывод замеров.
4. Действие 5:
    a. Проверка очереди массива.
    b. Добавления элемента в очередь массив.
5. Действие 6:
    a. Проверка очереди списка.
    b. Добавления элемента в очередь списка.
6. Действие 7:
    a. Проверка очереди массива.
    b. Удаление элемента из очереди массива.
7. Действие 8:
    a. Проверка очереди списка.
    b. Удаление элемента из очереди списка.
    c. Сохранение освобожденного адреса.
8. Действие 9:
    a. Проверка освобожденных адресов.
    b. Вывод освобожденных адресов.
9. Действие 0:
    a. Выход из программы.

НАБОР ТЕСТОВ

Название	Входные	Выходные	Результат
Некорректный ввод: вероятности	-1	-	ERR_IO: Ошибка ввода.
Некорректный ввод: вероятности	a1	-	ERR_IO: Ошибка ввода.
Некорректный ввод: интервала времени, одинаковые границы	1.0 1.0	-	ERR_IO: Ошибка ввода.
Некорректный ввод интервала времени,	2.0 1.0	-	ERR_IO: Ошибка ввода.

интервал не существует			
Некорректный ввод: отрицательное время	-1 2.0	-	ERR_IO: Ошибка ввода.
Некорректный ввод: Символ вместо времени	a 1.0	-	ERR_IO: Ошибка ввода.
Некорректный ввод: Кол-во элементов отрицательное	-100	-	ERR_IO: Ошибка ввода.
Некорректный ввод: Символ вместо кол-ва элементов	a	-	ERR_IO: Ошибка ввода
Некорректный ввод: в очередь добавляют символ	a	-	ERR_IO: Ошибка ввода
Корректный ввод параметров для модуляции очереди массива			
Корректный ввод параметров для модуляции очереди списка			
Корректный ввод числа, добавляемого в очередь массив	5		
Корректный ввод числа, добавляемого в очередь список	6		
Корректный ввод параметров для модуляции очереди массива, вероятность равна 1			

МЕТОДИКА ЗАМЕРОВ

Вычисление замеров добавления или удаления n элементов проводилось следующим образом: для каждого очереди находилось среднее за 100 замеров.

ЗАМЕРЫ

N, кол-во элементов	Удаление из очереди массива, мс	Добавление в очереди массива, мс	Удаление из очереди списка, мс	Добавление в очереди списка, мс
100	0.002230	0.002860	0.004960	0.005010
200	0.003030	0.003820	0.007400	0.008060
300	0.004010	0.005160	0.010500	0.010890
400	0.003070	0.003840	0.015800	0.011010
500	0.003200	0.004270	0.013800	0.009590
600	0.006330	0.008510	0.018130	0.019350

700	0.007020	0.009290	0.020410	0.021740
800	0.004160	0.005690	0.012380	0.013130
900	0.009850	0.013130	0.029470	0.030340
1000	0.005670	0.007820	0.017100	0.018230

Из замеров можно сделать вывод, что реализация первым способом среднем в 1,96 быстрее при добавлении и в 2,08 быстрее при удалении.

ПАМЯТЬ

N, кол-во элементов	Размер очереди массива в байтах.	Размер очереди списка в байтах.
100	80008	2416
200	80008	4816
300	80008	7216
400	80008	9616
500	80008	12016
600	80008	14416
700	80008	16816
800	80008	19216
900	80008	21616
1000	80008	24016

При работе с программой вовремя выполнения замеров и моделирования очереди списка фрагментация памяти не была обнаружена. Элементы типа sizeof(node_t) =16 располагаются друг за другом по адресам кратным их размерам.

ВЫВОДЫ

Наиболее эффективным по времени выполнений операций удаления и добавления является стек массива. В среднем первый способ в 1,96 раз быстрее при добавлении и в 2,08 при удалении за счет того, что ему нет необходимости выделять новые блоки памяти под добавленные элементы. Главным минусом очереди массива является выделение неиспользованной памяти. В случае списка задействуется вся необходимая память и у нее нет ограничений по размеру, в то время как в первом способе существует аварийная ситуация: переполнение.

В целом преимущества и недостатки двух реализаций очередей и стеков схожи. Операции над статическим массивом происходят гораздо эффективнее, в то время как список позволяет более гибко использовать память под процессы.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1.Что такое FIFO и LIFO?

FIFO (First In, First Out) и LIFO (Last In, First Out) — это два разных подхода к управлению данными в очередях и стеках соответственно. В очереди первый вошедший элемент выходит первым, а в стеке – последним.

2.Каким образом, и какой объем памяти выделяется под хранение очереди при различной ее реализации?

При реализации статическим массивом выделяется целый блок, размеры которого ограничены. В списке память выделяется динамически, что позволяет гибко управлять объемом памяти.

3.Каким образом освобождается память при удалении элемента из очереди при ее различной реализации?

При удалении в массиве достаточно лишь переместить указатель, в то время как в списке необходимо обновить указатель на следующий элемент и освободить память из-под текущего.

3.Что происходит с элементами очереди при ее просмотре?

Для просмотра очереди необходимо выделить дополнительную память (например: новую очередь), которая будет сохранять правильный порядок элементов в очереди при извлечении для повторного их добавления в ту же очередь.

4. От чего зависит эффективность физической реализации очереди?

Эффективность зависит от следующих факторов:

- Частота выполнения операций добавления/удаления элементов из очереди.
- Наличие фрагментации памяти.
- Выбор структуры данных для списка или массива.

6. Каковы достоинства и недостатки различных реализаций очереди в зависимости от выполняемых над ней операций?

Достоинства массива:

Быстрое выполнения операций за счет обращения по индексу.

Недостатки массива:

Ограничение по размеру.

Достоинства списка:

Нет ограничений по объему памяти.

Недостатки списка:

Малая эффективность выполнения операций из-за необходимости в освобождении памяти.

7. Что такое фрагментация памяти, и в какой части ОП она возникает?

Фрагментация памяти — это неэффективное использование памяти, при котором доступны небольшие свободные блоки, но они несмежные, что может приводить к невозможности выделения памяти для больших объектов. Возникает в куче.

8. Для чего нужен алгоритм «близнецов»?

Алгоритм "близнецов" применяется для управления сборкой мусора. Память делится на две одинаковые части. Активная копируется в неактивную, что гарантирует непрерывность и минимизацию фрагментации.

9. Какие дисциплины выделения памяти вы знаете?

- Выделение на стеке (статические структуры)
- Выделение в куче (динамические структуры)

10. На что необходимо обратить внимание при тестировании программы?

- Корректность работы алгоритмов.
- Управление выделением и освобождением памяти.
- Производительность и точность на краевых условиях.
- Обработка исключений и ошибок.

11. Каким образом физически выделяется и освобождается память при динамических запросах?

Это делается с помощью функций malloc(), calloc(), realloc() и free(). Системный аллокатор ищет подходящий свободный блок памяти в куче, управляет метаданными для учета используемой и свободной памяти и фрагментации.